Communication Messengers as a Basis for Distributed Algorithms*

Giovanna Di Marzo, Muhugusa Murhimanya, Christian F. Tschudin, David Billard, Jürgen Harms
Centre Universitaire d'Informatique, University of Geneva
e-mail: `dimarzo@cui.unige.ch`
http://cuiwww.unige.ch/tios/msgr/home.html

Abstract

The messenger paradigm is one of the earliest models which propose the exchange of code to implement computer communication. Code becomes mobile and mobile code is now considered a promising alternative for the implementation of distributed applications. One application of mobile code is in the implementation of software agents which themselves are used to implement complex distributed applications. However, efficient execution environments for mobile code are needed before mobile code can be considered a true technology for distributed applications. Our messenger project has focused at identifying and providing, both at the operating system level and at the language level, the necessary mechanisms for the efficient support of distributed applications built with messengers.

Since the project has reached a global consistent point, this paper, that can be considered as a position paper, provides an overview of the theoretical and practical aspects of the whole project. The main result of our work is the definition and implementation of an environment for the execution of messengers and the corresponding language for expressing the messenger behavior, called respectively the `MØ` platform and the `MØ` language. In both the language and the environment, minimality and efficiency have been the leitmotiv.

1 Introduction

Traditional approaches on mobile code focus on demonstrating "working applications" implemented with mobile code (the feasibility and viability of mobile code) and the fundamental work of efficient support of mobility is relegated to a second plan; *i.e.* the work of (a) identifying the different issues of mobility, (b) determining mechanisms that are needed to support efficiently mobility and (c) devising an efficient implementation for the different mechanisms identified in (b). This fundamental work has to be done at both the runtime environment level (platform level) and the language level.

Our research tries to avoid this "shortcut" and aims at exploring these fundamental issues. Our work is based on a small micro-kernel (platform for the execution of messengers) for supporting code mobility. In the micro-kernel, different mechanisms can be implemented and evaluated in respect to their effectiveness and pertinence to code mobility. At the same time, we focus on mechanisms at the language level by enhancing and extending the language used to express applications' behavior with mobile code. As a consequence, there is a constant feedback between the micro-kernel and the language; mechanisms identified at the language level are used to enhance the kernel and those identified at the kernel level can be evaluated to assess how they can support the language.

The results are a *distributed micro-kernel* to support mobile code (MOS) and the *underlying language* (`MØ`) to express mobile code behavior. Moreover, the need for new paradigms for expressing and implementing distributed applications with mobile code has been identified. We also focus on determining and developing an architecture for building distributed applications with mobile code.

Both theoretical and practical aspects of mobility are considered, the theoretical part of the work being concerned with deriving a formalization for messengers that can be used to reason about single messengers and families of messengers. In both parts, we make an effort [10, 11] to position our approach to code mobility with re-

*Published in the proceedings of the 2nd European Research Seminar on Advances in Distributed Systems (ERSADS'97), 1997 spect to similar efforts such as the Actor model [1] used for structuring distributed applications, the Linda/Polis paradigm [3, 5] used for coordinating applications through shared tuple spaces, but also to other machine-independent environments for mobile code, such as TACOMA [14], Obliq [2], Telescript [30], Java [12], and to mobile agents [13, 15, 29, 4].

The paper is organized as follows. Section 2 describes the messenger paradigm and sections 3 and 4 present the results of a two years work, classified from either their practical or theoretical aspects. Of course, the efforts pursued in the two parts aim at understanding and clarifying the same subject and are necessarily complementary. The last section concludes this paper.

2 The Messenger Paradigm

Messengers [21, 7] are mobile threads of execution useful for structuring distributed algorithms for both high- and low-level applications, *i.e.* computer communications [23] and distributed applications. The execution of a messenger takes place inside a *messenger platform.* Several messenger platforms are connected by unreliable channels through which messengers are sent as simple data packets.

Inside a given platform, messengers are *sequential processes* executing *concurrently and in parallel.* They *coordinate their execution* by the means of (1) a *shared dictionary,* and (2) *messenger queues.* The shared dictionary is a data structure accessible to all messengers and where a messenger can insert, change or remove data. A messenger is able to insert itself in a queue, its execution is then stopped until it reaches the head of the queue.

Messengers can *create at runtime* (1) new data, (2) new messengers in the platform where they are executing, and they can *move themselves* or *send other messengers* to other platforms. The communication between platform is restricted to the exchange of messengers, if data has to be sent to another platform, it is encapsulated in a messenger, which will retrieve the data from its code once it has arrived in the platform.

To summarize, messengers are anonymous and autonomous sequential processes, executing in parallel without central control, able to move between platforms and to cooperate and coordinate their work by the means of shared data structures and messenger queues. Messenger platforms enable messenger to execute and move freely between heterogeneous machines.

3 Practical aspects of Messengers

As said above, the main result of our work is (a) a distributed micro-kernel [25] called MOS (Messenger–Based Operating System) designed for supporting mobile code and (b) the underlying language called **MØ** [22], for expressing messenger behavior.

MOS

The fundamental features of MOS are high genericity and flexibility because no protocol are wired inside the micro-kernel. The basic idea is that the micro-kernel must provide only local services to messengers running on the platform. Messengers themselves cooperate for realizing services that span the network, *i.e.* services that require the collaboration of multiple nodes. Therefore, collaboration protocols needed to realize such services are shifted from the micro-kernel to the messengers[1]. Since there are no hard-wired protocols inside MOS, various operating systems can be implemented on top of the MOS micro-kernel. Currently an effort is pursued to emulate a UNIX operating system on top of MOS [19]. Moreover, it is possible to achieve interoperation between different OSs implemented on top of MOS without resorting to other gateway mechanisms.

The MOS micro-kernel provides only management of local resources which comprises: messenger creation and scheduling, local memory management, execution of native code, basic mechanisms to implement security policies and code execution on a remote node. A *currency mechanism* is used as a common and uniform way for managing the platform (MOS) resources [26]: messengers pay for resources they consume. Currently, this mechanism is applied to CPU-time, and memory. We plan to apply it to other resources, one adequate candidate being network bandwidth. Messengers interact with the micro-kernel on the node they are executing through the messenger language that is shared by all the nodes to express messenger behavior.

[1]A minimal set of messengers is provided, implementing basic communication protocols, synchronization, ...

Messenger-Based Services

Most distributed applications are structured under the client/server model that relies on the message-passing paradigm. Message-passing can be explicitly handled by the application or can be hidden in language level mechanisms, or runtime environments which translate high-level constructs in the exchange of messages. One such mechanism is the remote procedure call (RPC). The client/server model and the RPC mechanism seem overly restrictive for implementing distributed applications with mobile code. This is because they require static preconfiguration of entities (client and server) which exchange messages using a pre-established protocol. To benefit from the flexibility of mobile code, a more flexible architecture for distributed applications is currently investigated.

Indeed, we are developing an architecture for building distributed applications by composition of different services. This architecture allows the secure publication of services offered by service providers. In addition, it allows the clients (users of the service) to discover available published services and to determine which information is needed in order to interact with them. For that purpose, the user of the service determines at runtime the service's interface. This interface is described using the *messenger interface description language* that must be understood by both service providers and clients. We propose to identify ways to express "semantical", "operational" and "management" interfaces of distributed applications using a simple but extensible language.

Our messenger interface language (M-IDL) is a means comparable with the Interface Definition Language (IDL) [28] of CORBA. CORBA IDL and M-IDL have the same purpose, i.e. to help clients to use services offered by servers. However, CORBA IDL is based on the client/server paradigm and is a specification explaining how a server has to publish its services and how a client accesses them. M-IDL explains how a messenger can publish in its interface *the way how* other messengers can access its services and how other messengers can retrieve this information from the interface. CORBA advocates interface standardization to allow interoperability between heterogeneous software. Messengers execute in platforms which offer them an homogeneous environment in a network of hosts; focus is given on how a messenger, that only knows the name of a service, can nevertheless interact appropriately with it.

Dynamicity and Changing Environments

Till now, most distributed systems are rather static; dynamicity is not commonly handled automatically (e.g. manual shutdown of nodes). But this vision of distributed systems is now completely outdated with the new technology trends such as the advent of mobile (or nomadic) computing (e.g. laptops with wireless interfaces accessing fixed networks), where connection/disconnection or shutdown/startup are normal operations, since it is essential for these devices to save power or bandwidth.

Therefore, real messenger based distributed applications will execute in complex and dynamic environments where the available resources provided by the running nodes dynamically change, for example as the result of new nodes being booted and some nodes being shut down. In such environments, the flexibility of mobile code allows messengers to move through the network of nodes to search for resources and information they need to accomplish their task.

So it is now *required* that distributed applications be able to work smoothly in such dynamic environments. Therefore we should provide, in our messenger environment, mechanisms for managing system dynamicity. It is clear, following our philosophy of providing only local services at the MOS level, that management of system dynamicity has to be provided at the language level (messenger level) and not at the runtime level (the MOS level) because dynamicity management requires strong cooperation between different nodes.

Implementation

Currently, two messenger platforms have been implemented: (1) the **MØ** platform mentioned above, and (2) the MSGR-S platform. In the **MØ** platform, the messenger language, **MØ**, is based on PostScript. The **MØ** platform runs on both the Unix operating system and on MOS. In the MSGR-S platform [27], a functional approach is used and the messenger language is based on Scheme. Moreover, to allow users to experiment with messenger programming using a more comfortable language, a Pascal to **MØ** compiler, called PTOM, has been implemented [24].

Another important part of the project was the implementation of applications with the aim to determine mechanisms needed to support them efficiently. The implementation serves as a feedback to enhance both the messenger based operating system and the messenger language. We have implemented two basic services, namely a distributed semaphore service [17] and a distributed shared memory service. The distributed semaphore service allows messengers to synchronize their execution independently of their physical location; and the distributed shared memory service allows messengers running on different nodes to share information. We have developed a memory consistency model called Access Consistency [18] that supports mobility and which is well suited for the implementation of distributed shared memory in a messenger environment.

Open Questions

Clearly, the mobile code approach is a promising alternative for the implementation of distributed applications. Nevertheless, building distributed applications using the client/server model and the underlying message-passing mechanism is well understood and can be an attractive alternative. In some situations, the client/server approach can be more efficient, even though the mobile code approach can be more flexible. As a consequence, applications built using the mobile code approach have to coexist and certainly to interact with client/server based applications. This requires to determine and to implement an interface between messengers and the external world to them. More clearly, the following problems have to be solved: (a) how can applications built with messengers access the external world, and how can they coexist with other applications not necessarily expressed in terms of messengers? (b) how can other applications access services implemented by messengers?

4 Theoretical implications of Messengers

Messengers can be considered under different point of views: (1) a messenger can spread several other messengers through a network, therefore messengers can be viewed as concurrent distributed processes; (2) messengers can move themselves autonomously, thus they can be seen as mobile agents; (3) messengers interact with each other to realize their own goal or to participate in a global application, so they are considered under a coordination and compositional point of view.

Mobile Agent Theories and Coordination

A mathematical formalization of the messenger paradigm has been given in [6]. The devised operational semantics of the messenger paradigm, in terms of transition systems, will be useful for comparing messengers with other paradigms.

Features of messengers are mobility, creation of new messengers at runtime, interaction through a shared memory, no central control, composition of messengers into families and collaboration between messengers. An analysis and comparison of the messenger paradigm with similar paradigms, through a survey of existing theories can be found in [10], where we present paradigms and formalisms related to one or more of these features. This helps us to position the messenger paradigm among other existing paradigms. Messengers as mobile processes are investigated with π-calculus [16]; messengers as distributed concurrent processes are compared to actors; messengers as coordinated processes are compared to the PoliS paradigm; messengers as collaborative agents are investigated with temporal logic for agents. We also explain informally how two formalisms can be applied to messengers: (1) the π-calculus, which focuses on process mobility, and (2) the algebra for generative communication, which focuses on process coordination through a shared memory.

New formalisms related to mobility can be devised: (1) Petri nets and mobility: the idea is to allow tokens to be whole petri nets, a token which crosses a transition will cause a new net to appear; (2) Mix of higher-order π-calculus (HOπ) and algebra for generative communication: the idea is to use the mobility feature of HOπ (process passing), and the process coordination through shared memory of generative communication, to obtain a formalism exactly suited to messengers.

Once a formalism has been chosen, and has demonstrated to be well suited for messengers, it will be useful to provide a tool which translates messenger specifications into messenger programs.

In [9] we list what, at our sense, should be the components of a multi-agent system and give some hints on how elements of *category theory* can be applied to formalize these components.

PDU-based protocols and messenger-based protocols

A preliminary work towards a *translation of PDU-based protocols in messenger-based protocols* has been realized. Rules for translating protocols expressed with the exchange of messages in equivalent protocols expressed in terms of messengers have been proposed in [20]. The final goal is to implement an automated tool that can realize such transformations.

Open Questions

Messengers can work alone, but can also interact with each other through messenger queues and the shared dictionary. These interactions occur at runtime and without any central control. The behavior of the whole system results from the work of all messengers. Questions naturally arise: how to derive properties of the whole system? What is the global behavior of the whole system? How is it possible to extend a formalism for messengers to a formalism for families of messengers?

We have made the observation that most formalisms and paradigms related to parallel distributed applications share some common notions as: composition of components, encapsulation of the components' behavior, interface, emergence of properties, etc. We are interested in generalizing all these notions, inside a meta-formalism in order to derive the semantics of systems working with "components" (objects, agents, messengers, etc.) of some kind. Some ideas on how to realize a meta-formalism for composition which goes beyond the notions of processes, objects, messengers or agents, and which captures the semantics of distributed systems build with "components" are listed in [8].

5 Conclusion

In this paper, we have presented a general overview of the messenger project conducted at the University of Geneva. We gave emphasis on the results obtained after a two years period, the ongoing work and the future research we aim to achieve.

We grouped the results into practical, *i.e.* resulting from the design and implementation of the `MØ` platform and `MØ` language, and theoretical, *i.e.* results about the mathematical formalization of the messenger paradigm, the automated translation of PDU-based protocol into messengers-based protocol and the application of existing formalisms or theories to the formalization of messengers.

The ongoing work and the future research focus on families of messengers (inter-messengers relationships) for both the practical and theoretical aspects. In the practical part, we advocate for the development of an architecture for building messenger-based distributed applications, that have to be aware of the dynamic aspect of their environment and able to cooperate with traditional client/server applications. In the theoretical part, we have done a preliminary work to devise formalizations for reasoning about messengers (at the intra-messenger level) and show how they can be extended to cope with families of messengers (at the inter-messenger level).

References

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[2] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, January 1995.

[3] Nicholas Carriero and David Gelernter. LINDA in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[4] David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant agents for mobile computing. Technical Report RC 20010, IBM, T. J. Watson Research Center, Yorktown Heights, New York, March 1995.

[5] P. Ciancarini. Distributed Programming with Logic Tuple Spaces. Technical Report UBLCS-93-7, University of Bologna, 1993.

[6] G. Di Marzo, M. Muhugusa, and C. F. Tschudin. Mathematical Formalization of the Messenger Paradigm. Technical Report Cahier du CUI No 100, University of Geneva, 1996.

[7] G. Di Marzo, M. Muhugusa, C. F. Tschudin, and J. Harms. The Messenger Paradigm and its Impact on Distributed Systems. In Claus Unger and Ioan Alfred Letia, editors, *ICC'95 International Workshop on Intelligent Computer Communication*, pages 79–94, June 1995.

[8] G. Di Marzo, M. Muhugusa, C. F. Tschudin, and J. Harms. Formalization of Agents and Multi-Agent Systems. The Special Case of Category Theory - Working Paper. Technical Report Cahier du CUI No 109, University of Geneva, 1996.

[9] G. Di Marzo, M. Muhugusa, C. F. Tschudin, and J. Harms. Meta-Formalism for the Composition of Objects - Working Paper. Technical Report Cahier du CUI No 108, University of Geneva, 1996.

[10] G. Di Marzo, M. Muhugusa, C. F. Tschudin, and J. Harms. Survey of Theories for Mobile Agents. Technical Report Cahier du CUI No 106, University of Geneva, 1996.

[11] Giovanna Di Marzo, Murhimanya Muhugusa, and Christian F. Tschudin. *Agent Mobility*, chapter 20, pages 375–406. Sams.net, 1996. In Bots and other Internet Beasties. Joseph Williams.

[12] J. Gosling and H. McGilton. *The Java Language Environment: A White Paper*. Sun Microsystems, Inc.

[13] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile Agents: Are they a good idea? Technical report, IBM, T. J. Watson Research Center, Yorktown Heights, New York, March 1995.

[14] D. Johansen, R. van Renesse, and F. B. Schneider. An Introduction to the TACOMA Distributed System. Technical report, University of Tromso, June 1995.

[15] A. Lingnau and O. Drobnik. An Infrastructure for Mobile Agents: Requirements and Architecture. Technical report, University of Frankfurt, 1995.

[16] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes I and II. *Journal of Information and computation*, 100(1):1–40,41–77, 1992.

[17] M. Muhugusa, G. Di Marzo, C. F. Tschudin, and J. Harms. Distributed Semaphore in a Messenger Environment. In *Decentralized Intelligent and Multi-Agent Systems DIMAS 95*. Institute of Computer Science, AGH - Technical University of Mining and Metallurgy, Krakow, Poland, November 1995.

[18] M. Muhugusa, G. Di Marzo, C. Tschudin, and J. Harms. Access Consistency Memory Model for Messengers. Technical Report Cahier du CUI No 107, University of Geneva, 1996.

[19] Guy Neuschwander. Exécution du code natif dans l'environnement MOS. Master's thesis, University of Geneva, November 1996.

[20] Louis Armand Soavelo. Transformation de spécifications de protocoles en messagers. Master's thesis, University of Geneva, April 1996.

[21] C. F. Tschudin. *On the Structuring of Computer Communications*. PhD thesis, Université de Genève, 1993. Thèse No 2632.

[22] C. F. Tschudin. An Introduction to the MØ Messenger Language. Technical Report No 86 (Cahier du CUI), University of Geneva, 1994.

[23] C. F. Tschudin. Protokollimplementierung mit Kommunikationsboten. In *KiVS'95-Tagung, Chemnitz*, 1995.

[24] C. F. Tschudin. PTOM - A Pascal Translator for Mobile Code. Technical Report ifi-96.06, Institut für Informatik, University of Zürich, July 1996.

[25] C. F. Tschudin, G. Di Marzo, M. Muhugusa, and J. Harms. A distributed micro-kernel for communications messengers. Technical Report No 110 (Cahier du CUI), University of Geneva, 1996.

[26] Christian F. Tschudin. Open resource allocations for mobile code. In *Proceedings of Mobile Agents 97*, apr 1997.

[27] R. Lino Valverde. MSGR-S: Un environnement d'exécution de messagers basé sur un interpréteur Scheme parallèle. Diploma thesis, University of Geneva, 1994.

[28] Steve Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[29] P. Wayne. Agents away. *BYTE*, pages 118–133, May 1994.

[30] J. E. White. Telescript Technology: The Foundation for the Electronic Marketplace. White paper, General Magic, Inc., 2465 Latham Stree, Mountain View, CA 94040, 1994.